CatDetect, a framework for detecting Catalan tweets

Author: Sergi Plaza Cagigós
Supervisors: Francesc Solsona Tehàs, Jordi Vilaplana Mayoral

Abstract

This work deals with language detection. It includes new proposals ranging from lexicon and morphological analysis to an increasing use of machine learning solutions. In this case, the language study is focused on Catalan, a minority language. Difficulty even increases in detecting Catalan on tweets, messages written in the Twitter social network. To achieve that, a Twitter-Catalan corpus has been generated using lexicon and morphological approaches, which then will be used to create supervised models based on Machine Learning techniques. They are also evaluated in order to see which one obtains the best prediction score and thus, the best suitability to be used. The best model is to be used in a website, where users can test the algorithm interactively in a front-end webpage and in background by means of a webservice across a RESTful API.

Index Terms

Catalan, Language Detection, Twitter corpus, Machine Learning, website.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Language detection is the problem defined as the processing of the natural language in order to determine the language of a given sentence, paragraph or text [1]. The difficulty of natural language processing lies in the existence of more than six thousand different languages [2]. To achieve that, many different approaches have been proposed.

Catalan is a minority Language, so although language detection has been scarcely investigated, Catalan has not been widely studied [3]. Right now, even Twitter [4] does not tag Catalan tweets and it lacks of APIs [5] supporting Catalan detection.

This work presents two main contributions. The first one is based entirely on a lexicon analysis of the Catalan language. It looks for similarities in a corpus of tweets tagged manually as Catalan. The second one is devoted to find models by using supervised machine learning techniques [6]. Supervised methods are used to train and test a tagged corpus of Catalan tweets in order to find accurate models that guess the Catalan language for later tweets.

In the for a suitable machine learning approach, four different unsupervised models were used: *Naive-Bayes* [7], *Support Vector Machines* [8], *Decision Trees* [9] and a *Neural Network* [10]. In order to reach the maximum accuracy, a corpus containing a set of 1,500 already classified tweets is given to every machine to train and test their effectiveness.

As presented before, there is not known tools which solves the guess of Catalan language detection of tweets. Nonetheless, a lot of language detection algorithms can be found, but none of them gave us the proof about how they work, or what is behind the API which gives the service, and furthermore, none of them insight on lexicon nor grammar of tweets [11]. Given this problem, the first objective was to create a classified corpus to train and learn how the Catalan twitter language looks like. To achieve that, a morphological and lexicon analysis will be used using as proof everything which is already known about the Catalan language. Previously a pre-processing method will be used in every tweet to ensure that only the language-like inputs are passed in the analysis.

However, the creation of a classified corpus brings another problem. As said before, the Twitter API does not give the possibility to retrieve Catalan tweets, so the way used to get them mixed with tweets on other languages has been the retrieval of tweets from the area belonging to Barcelona. The way the tweets have been retrieved has been using the #eMovix project [12] database, which has the ability to retrieve tweets from all around the world. This way, we are sure that the machines will learn what tweet is Catalan, but most importantly, what is not.

The main objective of the project is to obtain a tool which is able, with almost no margin of error, to classify Catalan tweets and creating a werserver and an API that allow people to use it.

The paper is divided onto different sections. On section 2 some of the already existent detection language algorithms are looked into, in order to learn and retrieve some of the methods used to solve the problem. Furthermore, some already solved problems using machine learning techniques will also be discussed. Section 3 will approach every single method used to process the tweets from arrival to departure. On section 4, the testing results of the machine learning models are presented, giving a closer look to each one of them. Finally, section 6 will give a summary of the work presented and will take a look on things to come.

2 Related Work

In this section, examples of past research in language detection engines based on the pattern recognition/classification paradigm (i.e. Supervised Machine Learning) are presented and discussed. With the analysis of the research done before using ML methods, four of them will be tested to solve the same problem in order to see which one fits best.

Classifiers like Support Vector Machines (SVM) [8], Bayesian Networks (BN) [7] and Neural Networks (NN) [10] are largely used in language detection [1].

SVMs use geometric approaches to divide a multi-dimensional space into regions where a set of input values is associated with the same outcome. SVMs are of increasing interest and tend to perform well when put in situations of binary classification. SVMs have been successfully applied in language detection [13].

BNs are a probability-based inference model. This method has also been used in language detection. BNs have been in fact used to identify language of short texts [14], being a clearly approach at how tweets are written.

NNs are conceptual models that mathematically emulate the way neuronal tissue connects, learns, and makes decisions. Successful NN applications include both discriminating and temporal approaches to language identification [15].

Some other languages have been also studied. For example, *LinguaKit* [16] provides, among other features, language detection for Spanish and English. Moreover, a slightly different approach to language detection is used to detect multiple languages on a single document [17] using byte-level n-grams to classify instances.

3 METHODOLOGY

As the project is entirely focused on the Catalan language detection, the result made by the engine is a binary response: *Catalan* or *non-Catalan*. The technology used is presented first. Then, the main steps, preprocessing and lexicon analysis of the detection procedure are presented. The performance metrics are explained next. Finally, the webserver and API are introduced.

3.1 Technology

Python has been the selected language to implement the overall framework of this project. *Python* provides a large amount of data processing libraries. First of all, the programming language decision was about finding the right libraries to perform all the necessary processing and testing for the machine learning part, leaving the web service behind. The *NLTK* library [1] contains range of functions related to natural language processing. It provides lots of classes for data transformation and many others to implement the machine learning models as *Naive-Bayes*, among others. Another powerful and famous library used has been *SciKit Learn* [19], which provided more machine-learning libraries.

For the web-service implementation, in order to maintain sense with the rest of the project, the framework used was *Django* [20]. *Django* is the most famous *Python*-written framework for web application design which also provides its own *Django Rest Framework* to implement a RESTful API [21]. It also has its own front-end to manipulate the data.

3.2 Preprocessing

The preprocessing steps [22] are the ones which take the input data (the tweet) and analyze it in order to improve the detection rates of the algorithm. First of all, some heuristics are applied to the data corpus. Due to the complexity of the Catalan language, the data corpus must be as clean as possible, taking into account that most of the singularities the *Twitter* platform contains are presented as one of the biggest issues for the algorithm.

3.2.1 Preprocessing

Based on the pseudocode described in Algorithm 1, the steps taken for the preparation of the input data in making up the corpus, are the following:

- **Hashtags (#)** [23]
  Using basic regular expressions, any *hashtag* is removed, because most of the times they are in English or they do not provide any evidence about the language.
- **Mentions (@)** [24]
  By definition, a mention relates a user name, and thus, they are directly processed as a non Catalan word. This is very similar to the hashtag case.
- **URLs** [25]
  URLs are also removed from the input data as they cannot be treated and analyzed as language-unique elements.
- **Emoticons** [26]
  Emoticons are also elements which cannot be treated as language data. The input data is analyzed using a static array of emoticons which contains almost every combination of symbols used to express any kind of emotion.
- ***Emojis*** [27]
  In the same way as emoticons, emojis are UNICODE [28] symbols used to express emotions, feelings and actions. And so they are removed of the corpus.
  A large file containing every single *emoji* is used in order to find and remove them from the input data.
- **Uppercase letters, numbers and punctuation signs**
  All of them are removed from the data. Nevertheless, the uppercase letters are replaced with its equivalent as a lowercase letter, as the machine learning algorithms would treat uppercase and lowercase letters as different type of inputs.

Alg. 1 shows the Pseudocode of the preprocessing process, namely *Tweet Parser*. Basically, the algorithm receives the original input and passes it onto the first filter, which removes the *hashtags*. Then, the input without *hashtags* goes into the *mentions* filter. Successively the data goes through all the filters described previously. The outcome is a clean tweet. Some examples showing how the *Tweet Parser* algorithm works are shown in TABLE 1.

Tweet	**Preprocessed Tweet**
@marfarri hahaha!! No anava per tu.	hahaha no anava per tu
@AmadeuBrugues @arnauriwz @josepruana a clar! Però el @didaclopez t'empata.	a clar però el t'empata
Fantàstic... https://t.co/WQNC2zD2c8	fantàstic
¡ALARMANTE! Pablo Medina: Desapareció el libro que consta que Maduro no nació en el país https://t.co/9psyvavIFX RT @FELUZESPERANZA	alarmante pablo medina desapareció el libro que consta que maduro no nació en el país

TABLE 1: Examples of preprocessed *tweets*.

3.3 Lexicon analysis

Once the corpus has been cleaned by the *Tweet parser*, the detection algorithm assumes that every clean tweet is a potential Catalan-item. Therefore, some detection techniques described in this section are applied.

Algorithm 1 Tweet Parser.

function parsetweet

```
for each tweet in input do
    tweet = remove_hashtags(tweet)
    tweet = remove_mentions(tweet)
    tweet = remove_urls(tweet)
    tweet = remove_emoticons(tweet)
    tweet = remove_rt(tweet)
    tweet = tweet.lower()
    tweet = ''.join(n for n in tweet if not n.isdigit())
    tweet = ''.join(l for l in tweet if (l not in string.punctuation or l == '''))
    tweet = remove_emojis(tweet, emojis)
end for
```

end function

3.3.1 Words

The first step of the lexicon analysis goes through the *Freeling* framework. We will use the potential of this morphological analyzer in order to apply *stemming* [1] to all the remaining words in the input data as follows:

- Each word of the corpus is passed as argument to a stemming function (*find_lemma*) which returns either the stem of the word or the same word itself. The second case happens when the framework can not find the stem.
- The word received by the framework is compared with the one passed to the stemming function. If the received word is different from the one sent, it is assumed that a stem for the word has been found, thus counting as a potential Catalan word.

The second step uses a dictionary-based methodology. Using a list of 738 Catalan stopwords[2], the tweet is processed in order to find them. Although stopwords give no real sense to the sentence, it is found, as well as in many other languages, that they are the most frequent words used, so finding and cataloguing them as Catalan is of utmost importance to tag the tweet as Catalan.
According to this analysis, the Score (S_1) of the overall analyzed corpus is obtained by using the equation 1.

$$S_1 = \frac{s + j}{w} \tag{1}$$

Where $\boldsymbol{s}$ is the number of stopwords found, $\boldsymbol{j}$ is the number of stems found and $\boldsymbol{w}$ is the number of words in the input data. Algorithm 2 shows the pseudocode for the word analysis. The algorithm basically goes through the steps defined above. First, it deals with every word of the tweet and searches it on the stopwords list. If it is found, the stopwords counter increases by 1 and the word is removed from the input. Then, the remaining words of the inputs are passed onto the *Freeling* framework to find the stem. As before, if it is found, the stems counter is increased one unit and the word is removed from the input data.

3.3.2 Trigrams

[29] The final step is to analyze all the input data to find *trigrams*[3]. Given a file with 299 trigrams, the corpus is also analyzed in order to obtaining a second score (S_2) (equation 2). Alg. 3 shows

1. stemming. Process carried out in order to find the stem of a word.
2. stopwords. Words with high frequency of occurrence, but with no real context or meaning.
3. trigram. combination of 3 characters which has a potential chance of appearing in a sentence written in a given language. Appendix A [7] contains the list of trigrams for the Catalan language.

Algorithm 2 Word Detector.

```
function detect_words
  for each tweet in input do
    for each word in tweet do
      lemma = find_lemma(word)
      if lemma != word then
        lemmas += 1
        remove_word
      end if
      if word in stopwords then
        stopwords += 1
        remove_word
      end if
    end for
  end for
end function
```

the Pseudocode of the trigram analysis, which given the list. Every trigram is searched inside the input string in order to find it. If found, it adds 3 to the trigram counter.

$$S_2 = \frac{t}{l}, \tag{2}$$

where t is the number of characters matching a trigram (so when a trigram is found, this value increases by 3) and l is the number of characters of the input data. That gives sometimes a value greater than 1 because some characters belongs to more than one trigram at once.

The final score (S_3), defined in equation 3 is a combination of (S_1) and (S_2). That is, its product.

$$S_3 = S_1 \cdot S_2 \tag{3}$$

Algorithm 3 Trigram Detector.

```
function detect_trigrams
  for each trigram in trigrams do
    if trigram in tweet then
      trigrams += 3
    end if
  end for
end function
```

3.4 Performance Metrics

Bearing of every classifier was evaluated by its Precision (4) and Recall (5), whilst F-score (6) is also noted but it is considered an average between the first two, as it is defined as the *harmonic mean* between them.

Precision (formula 4) measures the number of verified true positives within all of the positive predictions made by the engine -in this case, verified Catalan or non-Catalan positives among all language guessings of tweets. On the other hand, *Recall* (formula 5) takes care about how many correct choices the model has guessed from all the possible choices. *tp* (true positives) are the number of correctly predicted positive tweets, fp (false positives) are the number of positive predicted tweets which were negative, and fn (false negatives) are the number of tweets which where positive but where predicted as negative by the engine.

$$Precision = \frac{tp}{tp + fp} \tag{4}$$

$$Recall = \frac{tp}{tp + fn} \tag{5}$$

$$FScore = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{6}$$

3.5 Webserver and API

Next step was to upload the framework of the engine responsible for making decisions into a webserver made up by a website *front-end* allowing users sending tweets to the engine in foreground and interactively and an API, thought for sending requests to the engine remotely via the APIRest webservice, in a background and batch mode by means of a local program. Answers are delivered in a *JSON* [31] format.

Fig. 1: Website front-end and API.

Figure 1 shows the interface design of the described website, which consists of a single input where the tweet are submitted by the front-end screen interface. The answer of the engine will be one of two boolean responses: Catalan or Not Catalan.

The API is located in the bottom of the webpage. It consists of a single *POST* call which receives the tweet text and returns a *JSON* response with the tweet, the classification result -'1' or '0' meaning Catalan or not-Catalan respectively and the timestamp of the query (see Example 3.1).

Example 3.1. Calling the API this way:

```
curl -X POST -F "text=hola que tal com estas"
http://127.0.0.1:8000/catdetect/detect/
```

Would return the following JSON:

```
"model": "catdetect.tweet",
"pk": 39,
"fields":
{
"text": "hola que tal com estas",
"classification": 1,
```

"pub_date": "2017-08-28T17:30:28.148Z"
}

4 RESULTS

Four different models were tested and evaluated to find the best results. These were Naïve Bayes [7], Support Vector Machine [8], Decision Tree [9] and Neural Networks [10].

In order to get the most accurate results for every model, the same corpus has been used in all of them, meaning that each model has received the same exact training and it has been tested with the same tweet-corpus. To achieve that, a file containing 1,500 already classified tweets is used in every execution, using the first three quarters of the file to train the models and the least quarter to test them.

In addition, regarding accuracy, the same environment has been used to test the models. Every test was performed on a virtual machine running *Ubuntu Desktop 17.04* [30] with 2 gigabytes of RAM -which is the 25 per cent of the total RAM of the host running the virtual machine- and two of the total four cores of the CPU.

4.1 Naive-Bayes

Naive-Bayes is one of many probabilistic classifiers [32] which bases on applying the bayesian theorem [33] assuming no dependencies between each feature. That means that the value assigned to one of the features is completely independent from the value of any of the other features. In our case, a Catalan sentence is a combination of words -features-. Each word contributes independently to the probability of the whole sentence to be Catalan, and it does not matter if some feature has appeared before, or which way it has been tagged.

Given an instance to be classified -in this case, a sentence- represented by the vector x on equation 7, where n defines every feature -word-, the model assigns to this sentence the probabilities shown in equation 8.

$$x = (x_1, ..., x_n) \tag{7}$$

$$p(C_k|x_1, ..., x_n) \tag{8}$$

Table 2 shows the performance results (Precision, Recall and F-score) obtained with the execution of the *Naive-Bayes* algorithm.

Naive-Bayes	Precision	Recall	F-score
Catalan	87%	99%	92%
non-Catalan	98%	80%	88%
Total	93%	90%	90%

TABLE 2: Naive Bayes evaluation outcomes.

Taking a first look at the algorithm statistics, we are able to find a good balance between the results of the Catalan and the non-Catalan evaluations. For the first one, Precision drops a little -meaning only 87% of the evaluations were correct-, but Recall result informs that 99% of the evaluated tweets marked as Catalan were, in fact, Catalan. On the other hand, the non-Catalan evaluation is 98% precise but 20% of the evaluated tweets were incorrectly labeled. That give us a final F-score of 90%, which probably makes the *Naive-Bayes* model the one which suits best to the problem.

4.2 Support Vector Machine

The *Support Vector Machine* -from now on SVM- model divides the input data into different classes, thus making SVM a non-probabilistic classifier [32].

In the case of SVM, data is represented as a *p*-dimensional vector -a list of *p* numbers-, and the goal is to look for an optimal (*p-1*)dimensional *hyperplane* -a plane with a *p-1* dimension-, which is the one with the largest distance to the closer point of each class.

In our case, data is classified in a 2-dimensional plane which is divided into two by the (2-1)dimensional hyperplane: a line which separates the theoretically Catalan and non-Catalan tweets. For a better understanding, figure 2 shows an example of data division of the algorithm, where the white and black dots represent the two different classes and the red vector defines the hyperplane which separates them.

Table 3 shows the performance results (Precision, Recall and F-score) obtained with the execution of the SVM algorithm.

SVM	Precision	Recall	F-score
Catalan	86%	98%	91%
non-Catalan	97%	81%	88%
Total	91%	90%	90%

TABLE 3: Support Vector Machine evaluation outcomes.

The SVM also finds its place in the final results, as none of the scores show an effectiveness of less than 80%. Nevertheless, a pattern arises between this results and the ones obtained in the *Naive-Bayes* evaluation shown on table 2. For the Catalan instances, Precision drops but Recall grows to 97% (almost 100%). However, the non-Catalan instances show very high Precision but Recall unveils a 10% of mistakes. At the end, the averages say the final score for the SVM algorithm also goes up to 90%, which makes the model competitive too.

4.3 Decision Tree

Decision trees build classification models based on a tree structure, which takes the dataset used as training and breaks it into smaller pieces in order to create subsets which result in a tree with *decision nodes* and *leaf nodes*. *Decision nodes* are the ones where the data gets split onto two or more decision nodes which again, will be split again until the final node -*leaf node*- is reached, where the result is found.

The easiest way to understand decision trees is to think about a set of *if* clauses -or *decision rules* [34]-. For example, for a Catalan set of words "*hola que tal*" the decision tree would ask:

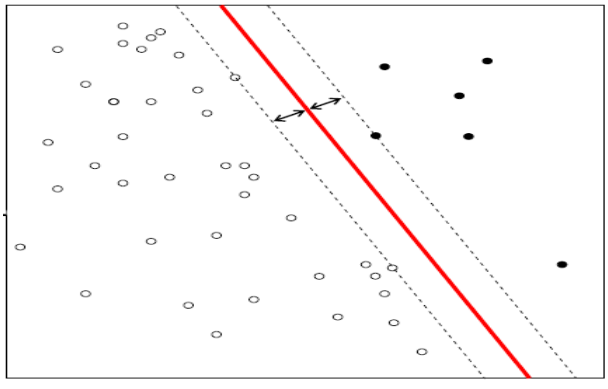

Fig. 2: Example of data division using a SVM model

if *hola* ***and*** *que* ***and*** *tal* ***then*** *CATALAN*

The question has to be understood by interpreting the *if* clauses as *decision nodes* which lead to the next *if* clause -the next *decision node*- or the *then* clause, which is interpreted as the *leaf node*, where the outcome results.

Table 4 shows the performance results (Precision, Recall and F-score) obtained with the execution of the *Decision Tree* algorithm.

Decision Tree	Precision	Recall	F-score
Catalan	91%	58%	67%
non-Catalan	72%	99%	84%
Total	83%	78%	76%

TABLE 4: Decision Tree evaluation outcomes.

On the first look to the results table, a significant drop in both Precision and Recall is found. On the Catalan instances side, Precision is rapidly discarded by a disappointing 58% Recall which tells us 42% of the evaluations were incorrectly labeled. Moreover, the non-Catalan instances show a Recall of 99%, showing us that almost every instance was correctly labeled but the Precision almost dropped below 70%. The average F-score for all instances is below 80%, which on some cases may not be a bad score, but here leaves decision trees out of the question.

4.4 Neural Network

Neural Network -NN from now on- is the model which goes beyond the probabilistic and the non probabilistic methods, as they try to simulate the way biological nervous systems work. If we take a look at a brain we see that it is composed by a large number of neurons which work together in order to solve a problem. In the learning field, neurons work with the adjustment of their synapses -connections- in order to learn from one another and solve the problem. The same happens with NN.

NN are usually divided in many layers, the first one being called *input* and the last one *output*. Between them, data suffers from many transformations while trespassing the different middle layers, taking into account that data always goes forward, so it always trespasses the layers in the same direction, and it never trespasses a layer twice.

Neurons work with weights. Every neuron will assign a weight to each and everyone of the incoming synapses. Every time the neuron receives new data from a synapse -a new weight- it multiplies it by the associated weight, and then every weight is added, resulting in a single final weight. If that number is below a threshold, the neuron does not send any data, but if the number goes beyond the threshold, it sends the number via all the synapses.

Table 5 shows the performance results (Precision, Recall and F-score) obtained with the execution of the *Neural Network* algorithm.

Neural Network	Precision	Recall	F-score
Catalan	78%	69%	73%
non-Catalan	73%	89%	80%
Total	76%	80%	78%

TABLE 5: Neural Network evaluation outcomes.

Although the results are fairly successful, timing has been a negative point for the NN. Training and testing of the NN with the 1,500 tweet corpus triggered more than 3 hours to complete. Leaving that behind, as the NN would be already trained, results are also disappointing as the total F-score has not gone over 80%. Both of the instances show a Recall below 90%, which makes the NN a non-reliable model against all other.

5 Discussion

As was reviewed, the *Naive-Bayes* model obtained the best score (90%). Taking a look at how the model works, and also at the features of the *corpus* used for training, it is safe to say, especially for longer tweets, that the model can successfully handle *missing data*. This means that if the tweet contains a word the model has never seen before, as it has low impact to the whole input, that word can be eliminated from the input and so it does not have a negative impact on the total score of the corpus.

As seen, *SVM* is a direct competitor against *Naive-Bayes* because the *corpus* only contains two classes. Thus, it is easy for the *SVM* model to create the function which separates the data into two clearly different sections. However, one of the main reasons for the selection of the *Naive-Bayes* model against SVM, is the easiness it provides in order to keep feeding the model with new training -to be discussed in the Future Work section 6- as it does not have the need to rebuild the whole model to gain information of the new incoming data.

Taking a look at *Decision Trees*, one of the reasons which would explain the lowest score obtained should be the large amount of instances which the tree receives for training. As the discovery of a new word -instance- has the ability to create one new branch with one new answer, hence making that instance of utmost importance when that makes no sense at all. However, that could be approached by the *pruning* technique, which reduces the size of the final tree by removing branches which provide the less decision power to the final classification, thus reducing complexity and increasing accuracy.

Finally, *NN* are a type of model which never fully-train a given a set of corpus, as the model keeps training by making connections and sharing knowledge between the neurons making up the network. On the other hand, one of the main features that *NN* has not been able to reach yet is what makes the human brain unique, to deduce multiple answers for the same question. *NN* have the ability to store multiple answers, but yet it will always answer the same as it bases on the weights discussed on section 4.4 and will always select the one which has the best weight of them all. The model took training for about 5 hours. We considered this a good time limit. However, training can continue for hours. It achieved a score higher than a 70%, but it is still left to future work investigating the training time insight on performance (score).

6 Conclusions and Future Work

We presented a methodology for the training and testing of a Tweet-Catalan corpus resulting in a fairly successful algorithm to detect Catalan using lexicon and morphological methods. Looking at the outcomes from this method, machine learning was put into doubt as the possibilities for it to achieve higher scores were rather low. However, after giving it a thought, ML provides a vast spectrum of techniques, each one tackling the problem from a totally different angle. When tested, some of the methods did not worked as well as they were supposed to as discussed before, but some others did. The idea from now on is to use the *Naive-Bayes* method to keep improving the classifier, both on the training technique and on the final use of the algorithm.

As for the web service, the first idea was to implement it using a *Spring* back-end and an *Angular JS* front-end, which both provide the latest state of the art technologies, but the other possibility was to use *Django* which provides the back and the front-end on a single application, and is as well as the algorithm for the classifier, written in *Python*. Due to the design and user-friendly reasons, the second was the one finally chosen.

Improvements of the corpus generation could be obtained by using the most frequent languages of the world by understanding which are the words or n-grams which appear both in Catalan and other languages. With such an information, it could be possible to mark some of the inputs as *neutral* instead of what the algorithm is doing now. That will probably improve

performance. Now, the training corpus only contains tweets in Catalan, Spanish and English. In order to improve even more the performance, the corpus has to be able to train the machine as many languages as possible.

Another aspect which would make F-score going up would be the use of synonyms. In the corpus generation process, multiple words could be automatically transformed into the same word. So later on, when the models train with such a corpus, the vocabulary of words would then be much more reduced than before, making it a lot easier for the models to guess.

Although *Naive-Bayes* has obtained the highest score of all four models, it could be possible to implement some sort of *Principal Component Analysis*, which would make linear combinations of instances to reduce complexity and would increase both effectiveness and efficiency.

Regarding the web service, it should be interesting to implement some kind of *feedback*, so the engine should improve the information of the input data every time it makes a new guessing. That way, the model would keep training and improving itself.

REFERENCES

[1] STEVEN BIRD, EWAN KLEIN, AND EDWARD LOPER *Natural Language Processing with Python. O'Reilly Media*. 2009.

[2] SIMONS, GARY F. AND CHARLES D. FENNIG *Ethnologue: Languages of the World, Twentieth edition. Dallas, Texas: SIL International*. 2017.

[3] BRUGUERA, JORDI *"Historia interna del catalán: léxico, formación de palabras y fraseología". In Ernst, Gerhard. Romanische Sprachgeschichte. 3. Berlin, New York: Walter de Gruyter. pp. 3045–3055*. 2008.

[4] LAURIE SEGALL, CNNMONEY *"Buy a vowel? How Twttr became Twitter"*. 2010.

[5] JACOBSON, DANIEL; WOODS, DAN; BRAIL, GREG *APIs a strategy guide. Sebastopol, Calif: O'Reilly Media. ISBN 978-1-4493-0892-6*. 2011.

[6] AFSHIN ROSTAMIZADEH, AMEET TALWALKAR *Foundations of Machine Learning, The MIT Press ISBN 9780262018258*. 2012.

[7] ZHANG, HARRY *The Optimality of Naive Bayes. FLAIRS2004 conference*. 2004.

[8] CRISTIANINI, NELLO; AND SHAWE-TAYLOR, JOHN *An Introduction to Support Vector Machines and other kernel-based learning methods, Cambridge University Press. ISBN 0-521-78019-5 (SVM Book)*. 2000.

[9] ROKACH, LIOR; MAIMON, O. *Data mining with decision trees: theory and applications. World Scientific Pub Co Inc. ISBN 978-9812771711*. 2008.

[10] RIPLEY, BRIAN D. *Pattern Recognition and Neural Networks. Cambridge University Press. ISBN 978-0-521-71770-0*. 2007.

[11] MALLARY JEAN TENORE *What Twitter teaches us about writing short & well. Poynter*. 2012.

[12] C. FEIXA, C. RUBIO, J. GANAU, F. SOLSONA. *L'Emigrant 2.0: emigració juvenil, nous moviments socials i xarxes digitals. – (Col·lecció Estudis ; 35), ISBN 9788439395348, 314.745.3-054.72–053.81(460:410.111)*. 2015.

[13] IVANA BALAZEVIC, MIKIO BRAUN, KLAUS-ROBERT MÜLLER *Language Detection For Short Text Messages In Social Media. arXiv:1608.08515 [cs.CL]*. 2016.

[14] FELA WINKELMOLEN, VIVIANA MASCARD *Statistical Language Identification of Short Texts. Department of Computer Science (DISI), University of Genova, Italy*.

[15] SEBASTIEN HERRY, CELESTIN SEDOGBO, BRUNO GAS, JEAN LUC ZARADER *Language Detection combining discriminating approach and temporal decision with neural network modeling. ISBN: 1-424400471-1*. 2006.

[16] CILENIS SL *Series of language tools. Continuous development*. 2017.

[17] MARCO LUI, JEY HAN LAU, TIMOTHY BALDWIN; DEPARTMENT OF COMPUTING AND INFORMATION SYSTEMS THE UNIVERSITY OF MELBOURNE; NICTA VICTORIA RESEARCH LABORATORY; DEPARTMENT OF PHILOSOPHY KING'S COLLEGE LONDON *Automatic Detection and Language Identification of Multilingual Documents*. 2014.

[18] KUCHLING, A. M. *Functional Programming HOWTO. Python v2.7.2 documentation. Python Software Foundation*. 2012.

[19] FABIAN PEDREGOSA; GAËL VAROQUAUX; ALEXANDRE GRAMFORT; VINCENT MICHEL; BERTRAND THIRION; OLIVIER GRISEL; MATHIEU BLONDEL; PETER PRETTENHOFER; RON WEISS; VINCENT DUBOURG; JAKE VANDERPLAS; ALEXANDRE PASSOS; DAVID COURNAPEAU; MATTHIEU PERROT; ÉDOUARD DUCHESNAY *Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 12: 2825–2830*. 2011.

[20] DJANGO *Django Official Documentation - Current and detailed documentation on nearly every aspect of Django.*

[21] RICHARDSON, LEONARD; AMUNDSEN, MIKE *RESTful Web APIs, O'Reilly Media, ISBN 978-1-449-35806-8*. 2013.

[22] S. KOTSIANTIS, D. KANELLOPOULOS, P. PINTELAS *"Data Preprocessing for Supervised Learning", International Journal of Computer Science, Vol 1 N. 2, pp 111–117*. 2006.

[23] RUSSELL BRANDOM *Who owns the hashtag? (It isn't Twitter)*. 2013.

[24] EVAN WILLIAMS *How @replies work on Twitter (and how they might). Twitter*. 2008.

[25] CURTIS, SOPHIE *"Twitter's t.co URL shortener used to spread spam*. 2015.

[26] YASUMOTO-NICOLSON, KEN *The History of Smiley Marks (English) Retrieved 2017-08-10*. 2007.

[27] HANNAH MILLER *Emoji Language of the Internet : Emojigraphy*. 2016.

[28] RICHARD GILLAM, ADDISON-WESLEY PROFESSIONAL *Unicode Demystified: A Practical Programmer's Guide to the Encoding Standard. 1st Edition. ISBN 0-201-70052-2*. 2002.

[29] CHRISTOPHER D. MANNING, HINRICH SCHÜTZE *Foundations of Statistical Natural Language Processing, MIT Press. ISBN 0-262-13360-1*. 1999.
[30] MAKO HILL, BENJAMIN; BACON, JONO; BURGER, COREY; JESSE, JONATHAN; KRSTIC, IVAN *The Official Ubuntu Book. p. 320. ISBN 0132435942*. 2006.
[31] ECMA INTERNATIONAL *The JSON Data Interchange Format*. 2013.
[32] HASTIE, TREVOR; TIBSHIRANI, ROBERT; FRIEDMAN, JEROME *The Elements of Statistical Learning p. 348*. 2009.
[33] LEE, PETER M *Bayesian Statistics: An Introduction, 4th edition. Wiley. ISBN 978-1-118-33257-3*. 2012.
[34] QUINLAN, J. R. *"Simplifying decision trees". International Journal of Man-Machine Studies. 27 (3): 221. doi:10.1016/S0020-7373(87)80053-6*. 1987.

7 APPENDIX A

Trigrams

- de
- es
- de
- la
- la
- el
- que
- el
- co
- ent
- s d
- qu
- i
- en
- er
- a
- ls
- nt
- pe
- e l
- a d
- en
- per
- ci
- ar
- ue
- al
- se
- est
- at
- es
- ts
- s
- pr
- aci
- un
- res
- men
- s e
- del
- s a
- s p
- re
- les
- l'
- na
- a l
- ca
- d'
- els
- a p
- ia
- ns
- con
- le
- tat
- a c
- i d
- a a
- ra
- a e
- no
- ant
- al
- t d
- s i
- di
- ta
- re
- a s
- com
- s c
- ita
- ons
- sta
- ica
- po
- r a
- in
- pro
- tre
- pa
- ues
- amb
- ion
- des
- un
- ma
- da
- s s
- a i
- an
- mb
- am
- l d
- e d
- va
- pre
- ter
- e e
- e c
- a m
- cia
- una
- i e
- nci
- tra
- te
- ona
- os
- t e
- n e
- l c
- ca
- cio
- l p
- tr
- par
- r l
- t a
- e p
- aqu
- nta
- so
- ame
- era
- r e
- e s
- ada
- n a
- s q
- si
- ha
- als
- tes
- va
- m
- ici
- nte
- s l
- s m
- i a
- or
- mo
- ist
- ect
- lit
- m s
- to
- ir
- a t
- esp
- ran
- str
- om
- l s
- st
- nts
- me
- no
- r d
- d'a
- l'a
- ats
- ria
- s t
- ta
- sen
- rs
- eix
- tar
- s n
- n l
- tal
- e a
- t p
- art
- mi
- ll
- tic
- ten
- ser
- aq
- ina
- ntr
- a f
- sti
- ol
- a q
- for
- ura
- ers
- ari
- int
- act
- l'e
- fi
- r s
- e t
- tor
- si
- ste
- rec
- a r
- fe
- is
- em
- n d
- car
- bre
- fo
- vi
- an
- ali
- i p
- ix
- ell
- l m
- pos
- orm
- l l
- i l
- ac
- fer
- s r
- ess
- eu
- e m
- ens
- ara
- eri
- sa
- ssi
- us
- ort
- tot
- ll
- por
- ora
- ci
- tan
- ass
- n c
- ost
- nes
- rac
- a u
- ver
- ont
- ha
- ti
- itz
- gra
- t c
- n
- a v
- ren
- cat
- nal

- ri
- qua
- t l
- do
- t s
- rma
- ual
- i s
- s f
- n p
- s v
- te
- t i
- ba
- cte
- tam
- man
- l t
- ial
- fa
- ic
- ve
- ble
- a n
- all
- tza
- ies
- le
- omp
- r c
- nc
- rti
- it
- rre
- fic
- any
- on
- sa
- r p
- tur